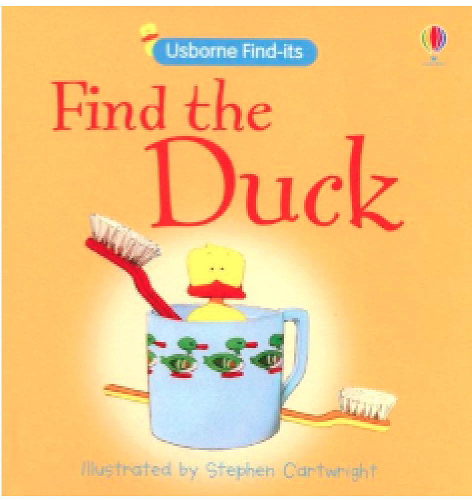
Open in Colab

# Tutorial: Object Detection with TFLite

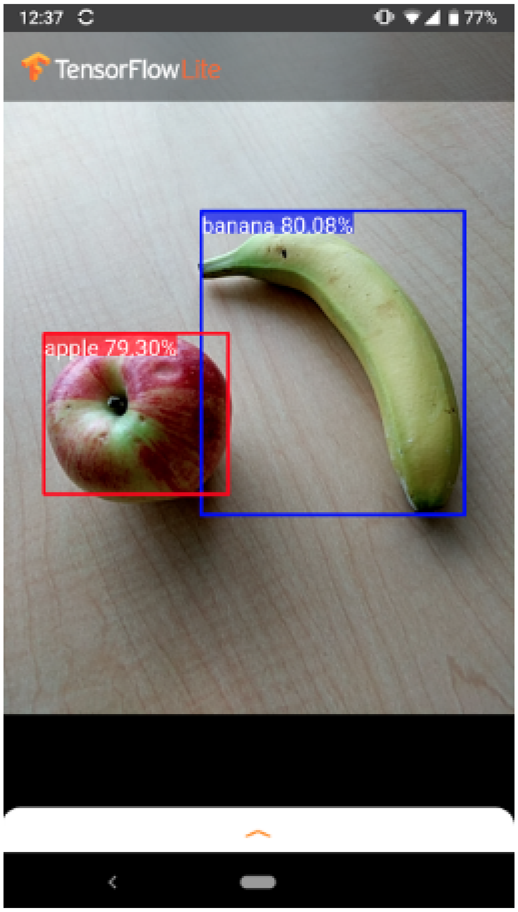
## Introduction

Imagine that you have a niece or a nephew and you want to give them a present. When you were growing up, your aunt gave you a "Find the Duck" book. You had lots of fun finding the duck on every page of this board book. Today, you want to make this book into a computer game. For that, you need to be able to teach the computer how to find the duck. This is what this tutorial will teach you.




## Our plan

The task that you are about to undertake is called "Object Detection." The good news is that the Google library called TensorFlow already does most of the groundwork for object detection. Furthermore, the TensorFlow Lite part of the library will help you to put your application on a phone or a device app. The end result of your object detection will look like a screenshot below, where you will be able to detect, out of a known set of objects, which ones are present in our picture and what are their locations.




We will do it in three steps. First, you will have to prepare the data: those objects that you will be looking to identify. After you got the objects, you will have to convert them to TFrecord format that Object Detection API expects. Then, you will train the model with this data. And finally, you will export the model to TFLite, preparing it to be used in your phone app. In the next tutorial, we will teach you how to use the resulting TFLite model in your phone app. So, let us start.

## Data collection

We have taken a dataset with pictures of fruit. Each data sample has an images of a fruit and an XML file with the fruit coordinates. This dataset came from Kaggle here, with a Creative Commons license, so for ease of use we have placed it in next to this tutorial.

In Jupyter Notebook, you can do command lines if you only start with a exclamation sign. Let us download this dataset first.

```
!wget -nc https://github.com/elephantscale/E2E-Object-Detection-in-TFLite/raw/master/
```

```
    File ‘Fruit_Images_for_Object_Detection.zip’ already there; not retrieving.
```

In the same way, let us unzip the dataset

```
!unzip -qqn Fruit_Images_for_Object_Detection.zip
```

## Generate intermediate files

To be able to generate TFRecords from our fruits dataset we first generate a `.csv` file that would contain the following fields -

- filename
- width
- height
- class
- xmin
- ymin
- xmax
- ymax

Now, we need to convert the XML descriptions to CSV. In your case, you may have to adjust this code, depending on the format of the data in your real-life dataset.

```
# Convert XML to CSV

import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'y
    xml_df = pd.DataFrame(xml_list, columns=column_name)
```

```
    return xml_df


def call_xml_to_csv():
    train = "/content/train_zip/train"
    test = "/content/test_zip/test"
    for directory in [train, test]:
        xml_df = xml_to_csv(directory)
        xml_df.to_csv('{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')


call_xml_to_csv()
```

```
Successfully converted xml to csv.
Successfully converted xml to csv.
```

```
!head -5 /content/train_zip/train_labels.csv
```

```
filename,width,height,class,xmin,ymin,xmax,ymax
banana_8.jpg,1920,1280,banana,496,168,1603,923
banana_8.jpg,1920,1280,banana,199,587,1470,1268
banana_8.jpg,1920,1280,banana,797,1,1772,611
banana_8.jpg,1920,1280,banana,1052,1,1916,550
```

```
!head -5 /content/test_zip/test_labels.csv
```

```
filename,width,height,class,xmin,ymin,xmax,ymax
banana_83.jpg,630,355,banana,63,17,560,323
orange_81.jpg,1300,990,orange,90,287,608,809
orange_81.jpg,1300,990,orange,560,217,1145,769
orange_81.jpg,1300,990,orange,323,37,786,551
```

Now that we have `.csv` files we can do some basic exploratory data analysis (EDA) to better understand the dataset.

## Basic EDA

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import cv2
import os


train_df = pd.read_csv("/content/train_zip/train_labels.csv")
test_df = pd.read_csv("/content/test_zip/test_labels.csv")
```

```
train_df.head()
```

| | filename | width | height | class | xmin | ymin | xmax | ymax |
|---|---|---|---|---|---|---|---|---|
| **0** | banana_8.jpg | 1920 | 1280 | banana | 496 | 168 | 1603 | 923 |
| **1** | banana_8.jpg | 1920 | 1280 | banana | 199 | 587 | 1470 | 1268 |
| **2** | banana_8.jpg | 1920 | 1280 | banana | 797 | 1 | 1772 | 611 |
| **3** | banana_8.jpg | 1920 | 1280 | banana | 1052 | 1 | 1916 | 550 |
| **4** | apple_31.jpg | 780 | 439 | apple | 304 | 105 | 773 | 439 |

```
test_df.head()
```

| | filename | width | height | class | xmin | ymin | xmax | ymax |
|---|---|---|---|---|---|---|---|---|
| **0** | banana_83.jpg | 630 | 355 | banana | 63 | 17 | 560 | 323 |
| **1** | orange_81.jpg | 1300 | 990 | orange | 90 | 287 | 608 | 809 |
| **2** | orange_81.jpg | 1300 | 990 | orange | 560 | 217 | 1145 | 769 |
| **3** | orange_81.jpg | 1300 | 990 | orange | 323 | 37 | 786 | 551 |
| **4** | orange_77.jpg | 732 | 549 | orange | 1 | 123 | 130 | 299 |

```
train_df["class"].value_counts()
```

```
banana    169
apple     156
orange    140
Name: class, dtype: int64
```

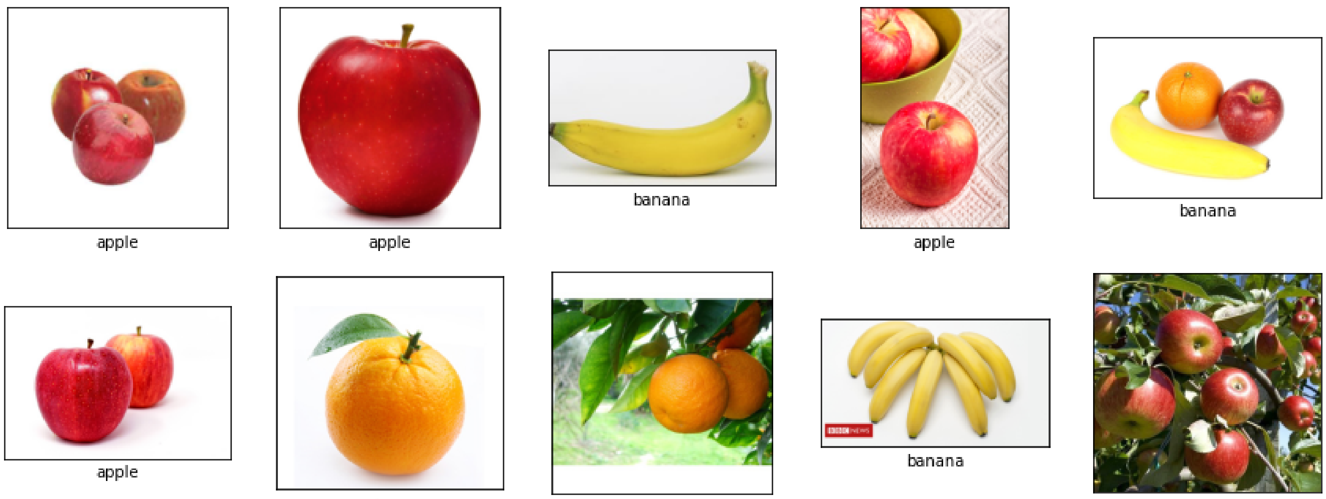
```
test_df["class"].value_counts()
```

```
orange    42
banana    40
apple     35
Name: class, dtype: int64
```

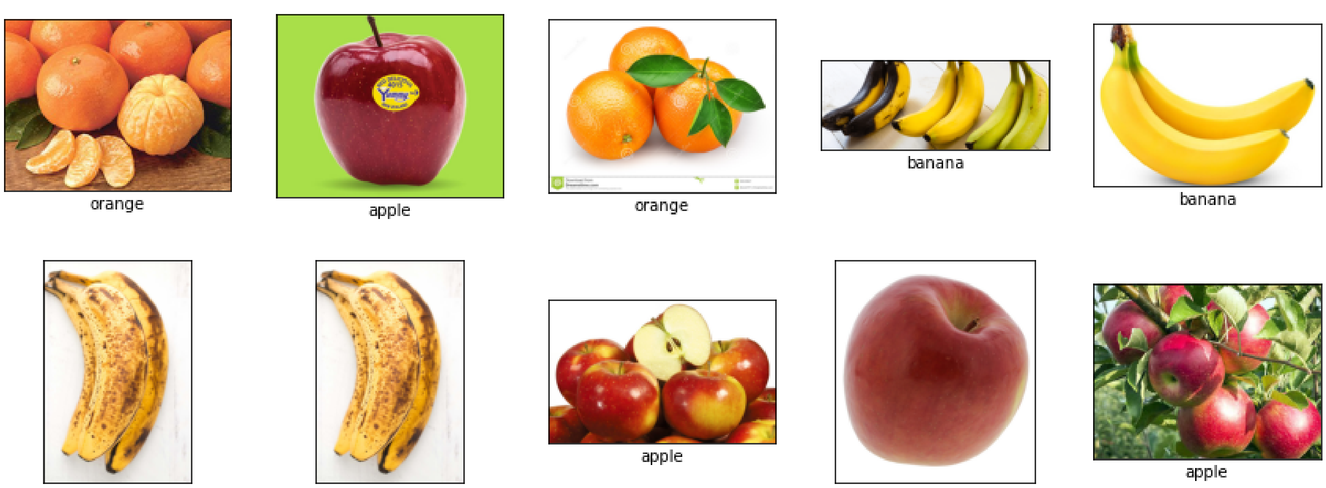
```
def show_images(df, is_train=True):
    if is_train:
        root = "/content/train_zip/train"
    else:
        root = "/content/test_zip/test"
    plt.figure(figsize=(15,15))
    for i in range(10):
        n = np.random.choice(df.shape[0], 1)
        plt.subplot(5,5,i+1)
        plt.xticks([])
```

```
        plt.yticks([])
        plt.grid(True)
        image = plt.imread(os.path.join(root, df["filename"][int(n)]))
        plt.imshow(image)
        label = df["class"][int(n)]
        plt.xlabel(label)
    plt.show()
```

```
show_images(train_df)
```



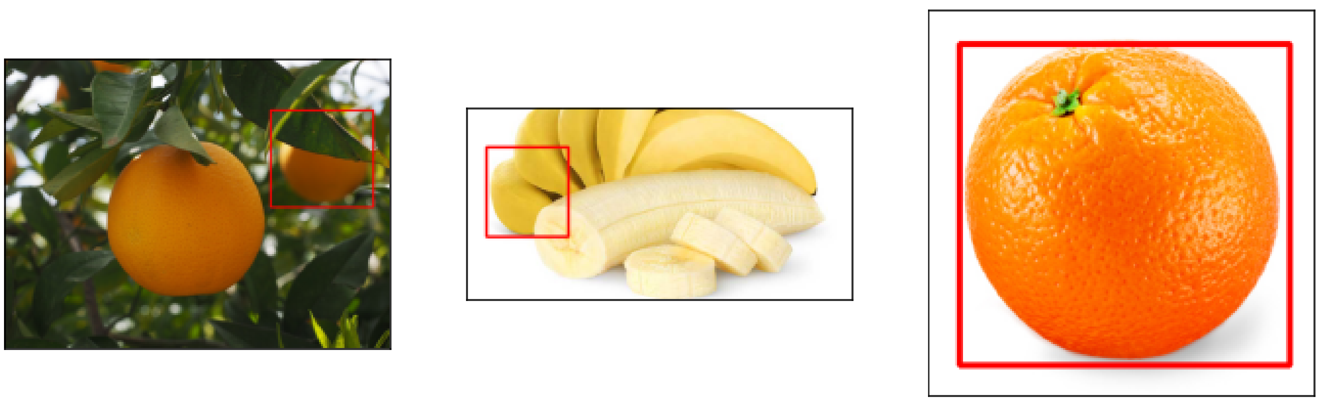

```
show_images(test_df, is_train=False)
```



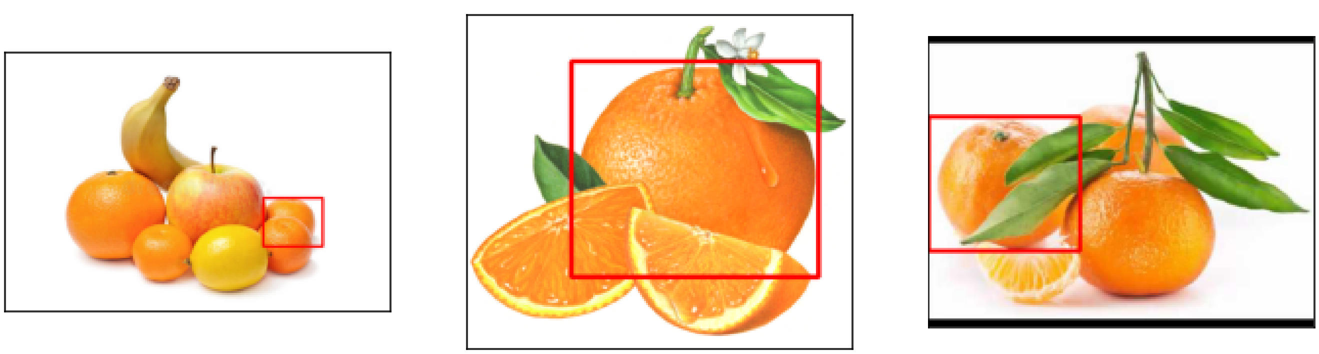

```
def verify_annotations(df, is_train=True):
```

```
    if is_train:
        root = "/content/train_zip/train"
    else:
        root = "/content/test_zip/test"

    plt.figure(figsize=(12,12))
    for i in range(3):
        n = np.random.choice(df.shape[0], 1)
        plt.subplot(1,3,i+1)
        plt.xticks([])
        plt.yticks([])

        image = plt.imread(os.path.join(root, df["filename"][int(n)]))
        xmin, ymin = int(df["xmin"][int(n)]), int(df["ymin"][int(n)])
        xmax, ymax = int(df["xmax"][int(n)]), int(df["ymax"][int(n)])
        cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (255,0,0), 3)
        plt.imshow(image)

    plt.show()
```

```
verify_annotations(train_df, is_train=True)
```



```
verify_annotations(test_df, is_train=False)
```



As we can see the dataset has annotation issues. So, our model training can suffer a lot from this.

## Generate TFRecords and `.pbtxt`

The TFRecord format is a simple format for storing a sequence of binary records. The format is explained here but for our purposes it is enough that the code below created these records for us.

The utility scripts that I used in the following cells were adapted from this repository.

TODO - should we convert to TF2 here?

```
%tensorflow_version 1.x
import tensorflow as tf
print(tf.__version__)

!git clone https://github.com/tensorflow/models.git

% cd models/research
!pip install --upgrade pip
# Compile protos.
!protoc object_detection/protos/*.proto --python_out=.
# Install TensorFlow Object Detection API.
!cp object_detection/packages/tf1/setup.py .
!python -m pip install --use-feature=2020-resolver .
```

```
1.15.2
Cloning into 'models'...
remote: Enumerating objects: 49553, done.
remote: Total 49553 (delta 0), reused 0 (delta 0), pack-reused 49553
Receiving objects: 100% (49553/49553), 558.66 MiB | 39.40 MiB/s, done.
Resolving deltas: 100% (34186/34186), done.
/content/models/research/models/research
Requirement already satisfied: pip in /usr/local/lib/python3.6/dist-packages (20
WARNING: --use-feature=2020-resolver no longer has any effect, since it is now t
Processing /content/models/research/models/research
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: lxml in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: contextlib2 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: tf-slim in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: pycocotools in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: lvis in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.1.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: pyparsing>=2.4.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: cycler>=0.10.0 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: opencv-python>=4.1.0.25 in /usr/local/lib/python3
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-pac
```

```
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: absl-py>=0.2.2 in /usr/local/lib/python3.6/dist-p
Building wheels for collected packages: object-detection
  Building wheel for object-detection (setup.py) ... done
  Created wheel for object-detection: filename=object_detection-0.1-py3-none-any
  Stored in directory: /tmp/pip-ephem-wheel-cache-12tt468k/wheels/ab/a8/ef/cead6
Successfully built object-detection
Installing collected packages: object-detection
  Attempting uninstall: object-detection
    Found existing installation: object-detection 0.1
    Uninstalling object-detection-0.1:
      Successfully uninstalled object-detection-0.1
Successfully installed object-detection-0.1
```

```
#!wget https://raw.githubusercontent.com/elephantscale/E2E-Object-Detection-in-TFLite
!wget https://raw.githubusercontent.com/elephantscale/E2E-Object-Detection-in-TFLite/
```

```
--2020-12-25 20:26:57--  https://raw.githubusercontent.com/elephantscale/E2E-Obj
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.13
HTTP request sent, awaiting response... 200 OK
Length: 3740 (3.7K) [text/plain]
Saving to: 'generate_tfrecord.py'

generate_tfrecord.p 100%[===================>]   3.65K  --.-KB/s    in 0s

2020-12-25 20:26:57 (74.7 MB/s) - 'generate_tfrecord.py' saved [3740/3740]
```

```
!python generate_tfrecord.py \
    --csv_input=/content/train_zip/train_labels.csv \
    --output_path=/content/train_zip/train.record
```

```
WARNING:tensorflow:From generate_tfrecord.py:107: The name tf.app.run is depreca

WARNING:tensorflow:From generate_tfrecord.py:88: The name tf.python_io.TFRecordW

W1225 20:26:59.525310 140510281471872 module_wrapper.py:139] From generate_tfrec

WARNING:tensorflow:From generate_tfrecord.py:47: The name tf.gfile.GFile is depr

W1225 20:26:59.573870 140510281471872 module_wrapper.py:139] From generate_tfrec

Successfully created the TFRecords: /content/train_zip/train.record
```

Before the running the cell below please edit the `path` variable in the `main()` function of `generate_tfrecord.py`. `generate_tfrecord.py` should be located here - `/content/models/research`.

```
!python generate_tfrecord.py \
```

```
!python generate_tfrecord.py \
    --csv_input=/content/test_zip/test_labels.csv \
    --output_path=/content/test_zip/test.record
```

```
WARNING:tensorflow:From generate_tfrecord.py:107: The name tf.app.run is depreca

WARNING:tensorflow:From generate_tfrecord.py:88: The name tf.python_io.TFRecordW

W1225 20:27:02.107374 140322520577920 module_wrapper.py:139] From generate_tfrec

WARNING:tensorflow:From generate_tfrecord.py:47: The name tf.gfile.GFile is depr

W1225 20:27:02.124306 140322520577920 module_wrapper.py:139] From generate_tfrec

Successfully created the TFRecords: /content/test_zip/test.record
```

```
!pwd
!ls -lh /content/test_zip/*.record
!ls -lh /content/train_zip/*.record
```

```
/content/models/research/models/research
-rw-r--r-- 1 root root 6.8M Dec 25 20:27 /content/test_zip/test.record
-rw-r--r-- 1 root root 23M Dec 25 20:27 /content/train_zip/train.record
```

Be sure to store these `.record` files to somewhere safe. Next, we need to generate a `.pbtxt` file that defines a mapping between our classes and integers. In the `generate_tfrecord.py` script, we used the following mapping -

```
def class_text_to_int(row_label):
    if row_label == 'orange':
        return 1
    elif row_label == 'banana':
        return 2
    elif row_label == 'apple':
        return 3
    else:
        return None
```

```
label_encodings = {
    "orange": 1,
    "banana": 2,
    "apple": 3
}

f = open("/content/label_map.pbtxt", "w")

for (k, v) in label_encodings.items():
```

```
    item = ("item {\n"
            "\tid: " + str(v) + "\n"
            "\tname: '" + k + "'\n"
            "}\n")
    f.write(item)

f.close()

!cat /content/label_map.pbtxt
```

```
item {
        id: 1
        name: 'orange'
}
item {
        id: 2
        name: 'banana'
}
item {
        id: 3
        name: 'apple'
}
```

Be sure to save this file as well. Next we will proceed toward training a custom detection model with what we have so far. Follow the steps in this notebook.

In this notebook we will be fine-tuning a **MobileDet** model on the **fruits dataset**. The original model checkpoints were generated in TensorFlow 1, so we need to stick to a TF 1 runtime. The purpose is to demonstrate the workflow here and not achieve state-of-the-art results. So, please expect unexpected performance for a shorter training schedule. Toward the very end, we will also see how to optimize our fine-tuned model using TensorFlow Lite APIs and run inference with it. This part will be executed on a TF 2 runtime.

As a prerequisite, you should be familiar with the contents of this notebook. It deals with the dataset contstruction part.

# Fetch pre-trained MobileDet model checkpoints and configuration

MobileDet comes in different variants (refer here). We will be using the `ssdlite_mobiledet_cpu` variant.

TFOD API operates with configuration files to train and evaluate models (the TF 2 release supports eager model execution too). For the purpose of this notebook, I created a configuration file following instructions from here. Note that I purposefully kept the `num_steps` argument to 2000. Here's a *non-exhaustive* list of the arguments I changed -

- `batch_size: 32`
- `label_map_path` and `input_path` inside `train_input_reader` and `tf_record_input_reader` respectively. The `num_examples` argument inside `eval_config` is set to 117.

## Download model checkpoint and config

```
!cd /content/models/research
!wget -q http://download.tensorflow.org/models/object_detection/ssdlite_mobiledet_cpu
!wget -q https://gist.githubusercontent.com/sayakpaul/9efad54dee957cc55b3adacf992a7a4
```

## Untar and verify the file structure of the model checkpoints

```
!cd /content/models/research
!tar -xvf ssdlite_mobiledet_cpu_320x320_coco_2020_05_19.tar.gz
!cp /content/label_map.pbtxt /content/models/research
!cp /content/test_zip/test.record /content/models/research
!cp /content/train_zip/train.record /content/models/research
```

```
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/model.ckpt-400000.data-00000-of-00
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/model.ckpt-400000.index
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/model.ckpt-400000.meta
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/tflite_graph.pbtxt
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/tflite_graph.pb
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/pipeline.config
ssdlite_mobiledet_cpu_320x320_coco_2020_05_19/model.tflite
```

# Model training

## Start training

**Note**: This script interleaves both training and evaluation. Before starting the training verify the paths carefully.

```
PIPELINE_CONFIG_PATH="/content/models/research/ssdlite_mobiledet_cpu_320x320_fruits_s
MODEL_DIR="/content/models/research/ssdlite_mobiledet_cpu_320x320_coco_2020_05_19"
```

```
!python object_detection/model_main.py \
    --pipeline_config_path={PIPELINE_CONFIG_PATH} \
    --model_dir={MODEL_DIR} \
    --alsologtostderr
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.52
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.58
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.57
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.59
INFO:tensorflow:Finished evaluation at 2020-12-25-20:26:00
I1225 20:26:00.477211 140098167961472 evaluation.py:275] Finished evaluation a
INFO:tensorflow:Saving dict for global step 2000: DetectionBoxes_Precision/mAP
I1225 20:26:00.477520 140098167961472 estimator.py:2049] Saving dict for globa
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 2000: /conten

I1225 20:26:00.480742 140098167961472 estimator.py:2109] Saving 'checkpoint_pa
INFO:tensorflow:Performing the final export in the end of training.
I1225 20:26:00.481420 140098167961472 exporter.py:410] Performing the final ex
INFO:tensorflow:Calling model_fn.
I1225 20:26:00.724067 140098167961472 estimator.py:1148] Calling model_fn.
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:03.809365 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:03.881179 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:03.954153 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:04.027084 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:04.103517 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:04.176319 140098167961472 convolutional_box_predictor.py:156] dept
INFO:tensorflow:Done calling model_fn.
I1225 20:26:04.843393 140098167961472 estimator.py:1150] Done calling model_fn
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/sa
Instructions for updating:
This function will only be available through the v1 compatibility library as t
W1225 20:26:04.843645 140098167961472 deprecation.py:323] From /tensorflow-1.1
Instructions for updating:
This function will only be available through the v1 compatibility library as t
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
I1225 20:26:04.844203 140098167961472 export_utils.py:170] Signatures INCLUDED
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
I1225 20:26:04.844320 140098167961472 export_utils.py:170] Signatures INCLUDED
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['tensorflow/servin
I1225 20:26:04.844394 140098167961472 export_utils.py:170] Signatures INCLUDED
INFO:tensorflow:Signatures INCLUDED in export for Train: None
I1225 20:26:04.844467 140098167961472 export_utils.py:170] Signatures INCLUDED
INFO:tensorflow:Signatures INCLUDED in export for Eval: None
I1225 20:26:04.844529 140098167961472 export_utils.py:170] Signatures INCLUDED
2020-12-25 20:26:04.845001: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:04.845482: I tensorflow/core/common_runtime/gpu/gpu_device.cc
name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
pciBusID: 0000:00:04.0
2020-12-25 20:26:04.845586: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845621: I tensorflow/stream_executor/platform/default/dso_
```

```
2020-12-25 20:26:04.845647: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845671: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845692: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845711: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845732: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:04.845827: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:04.846229: I tensorflow/stream_executor/cuda/cuda_gpu_executo
```

The above code block would take approximately **30 minutes** to run (although it depends on the GPU you got if you are running on Colab). If you increase the number of steps it would be even more. After the training was completed I got the following output -

```
I0915 04:48:33.129830 139851326252928 estimator.py:371] Loss for final step: 1.0553685.
```

# Export TFLite compatible graph

To export the fine-tuned checkpoints to a TFLite model we first need to export a model graph that is compatible with TFLite. More instructions about this are available here. First, we need to determine which checkpoints to be used to export the graph. Let's first take a look at our `MODEL_DIR` to get an idea.

```
!ls -lh $MODEL_DIR
```

```
total 300M
-rw-r--r-- 1 root   root   221 Dec 25 20:25 checkpoint
drwxr-xr-x 2 root   root  4.0K Dec 25 20:06 eval_0
-rw-r--r-- 1 root   root   27M Dec 25 20:25 events.out.tfevents.1608926126.0320a
drwxr-xr-x 3 root   root  4.0K Dec 25 20:26 export
-rw-r--r-- 1 root   root   17M Dec 25 19:55 graph.pbtxt
-rw-r--r-- 1 root   root   27M Dec 25 19:55 model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 root   root   36K Dec 25 19:55 model.ckpt-0.index
-rw-r--r-- 1 root   root  7.4M Dec 25 19:55 model.ckpt-0.meta
-rw-r--r-- 1 root   root   27M Dec 25 20:15 model.ckpt-1321.data-00000-of-00001
-rw-r--r-- 1 root   root   36K Dec 25 20:15 model.ckpt-1321.index
-rw-r--r-- 1 root   root  7.4M Dec 25 20:15 model.ckpt-1321.meta
-rw-r--r-- 1 root   root   27M Dec 25 20:25 model.ckpt-2000.data-00000-of-00001
-rw-r--r-- 1 root   root   36K Dec 25 20:25 model.ckpt-2000.index
-rw-r--r-- 1 root   root  7.4M Dec 25 20:25 model.ckpt-2000.meta
-rw-r----- 1 475825 89939  32M May 19  2020 model.ckpt-400000.data-00000-of-0000
-rw-r----- 1 475825 89939  14K May 19  2020 model.ckpt-400000.index
-rw-r----- 1 475825 89939 9.9M May 19  2020 model.ckpt-400000.meta
-rw-r--r-- 1 root   root   27M Dec 25 20:05 model.ckpt-653.data-00000-of-00001
-rw-r--r-- 1 root   root   36K Dec 25 20:05 model.ckpt-653.index
-rw-r--r-- 1 root   root  7.4M Dec 25 20:05 model.ckpt-653.meta
-rw-r----- 1 475825 89939  16M May 19  2020 model.tflite
-rw-r----- 1 475825 89939 4.6K May 19  2020 pipeline.config
```

```
-rw-r----- 1 475825 89939  17M May 19  2020 tflite_graph.pb
-rw-r----- 1 475825 89939  47M May 19  2020 tflite_graph.pbtxt
```

The checkpoint files with the prefix `model.ckpt-2000` are the ones we would be going with.

```
#Export TFLite compatible graph
#Always verify the paths before running this command.

!python object_detection/export_tflite_ssd_graph.py \
    --pipeline_config_path=$PIPELINE_CONFIG_PATH \
    --trained_checkpoint_prefix=$MODEL_DIR/model.ckpt-2000 \
    --output_directory=$MODEL_DIR \
    --add_postprocessing_op=true
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tf_slim/layers/
Instructions for updating:
Please use `layer.__call__` method instead.
W1225 20:26:11.416062 140442421712768 deprecation.py:323] From /usr/local/lib/
Instructions for updating:
Please use `layer.__call__` method instead.
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.184998 140442421712768 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.260566 140442421712768 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.334921 140442421712768 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.409476 140442421712768 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.488247 140442421712768 convolutional_box_predictor.py:156] dept
INFO:tensorflow:depth of additional conv before box predictor: 0
I1225 20:26:14.562946 140442421712768 convolutional_box_predictor.py:156] dept
2020-12-25 20:26:14.664051: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.699650: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.700199: I tensorflow/core/common_runtime/gpu/gpu_device.cc
name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
pciBusID: 0000:00:04.0
2020-12-25 20:26:14.700539: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.702338: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.703908: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.704223: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.714564: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.724804: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.735637: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.735765: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.736402: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.736906: I tensorflow/core/common_runtime/gpu/gpu_device.cc
2020-12-25 20:26:14.748542: I tensorflow/core/platform/profile_utils/cpu_utils
2020-12-25 20:26:14.748726: I tensorflow/compiler/xla/service/service.cc:168]
2020-12-25 20:26:14.748753: I tensorflow/compiler/xla/service/service.cc:176]
2020-12-25 20:26:14.860130: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.860796: I tensorflow/compiler/xla/service/service.cc:168]
2020-12-25 20:26:14.860831: I tensorflow/compiler/xla/service/service.cc:176]
```

```
2020-12-25 20:26:14.861010: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.861537: I tensorflow/core/common_runtime/gpu/gpu_device.cc
name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
pciBusID: 0000:00:04.0
2020-12-25 20:26:14.861622: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861654: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861680: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861709: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861736: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861760: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861783: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.861864: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.862433: I tensorflow/stream_executor/cuda/cuda_gpu_executo
2020-12-25 20:26:14.862904: I tensorflow/core/common_runtime/gpu/gpu_device.cc
2020-12-25 20:26:14.862971: I tensorflow/stream_executor/platform/default/dso_
2020-12-25 20:26:14.864113: I tensorflow/core/common_runtime/gpu/gpu_device.cc
2020-12-25 20:26:14.864150: I tensorflow/core/common_runtime/gpu/gpu_device.cc
2020-12-25 20:26:14.864165: I tensorflow/core/common_runtime/gpu/gpu_device.cc
2020-12-25 20:26:14.864317: I tensorflow/stream_executor/cuda/cuda_gpu_executo
```

## Verify the TFLite compatible graph size

It should have the `.pb` extension. Be sure to note down the path you would get as the output of code block.

```
!ls -lh $MODEL_DIR/*.pb
```

```
-rw-r----- 1 475825 89939 14M Dec 25 20:26 /content/models/research/ssdlite_mobi
```

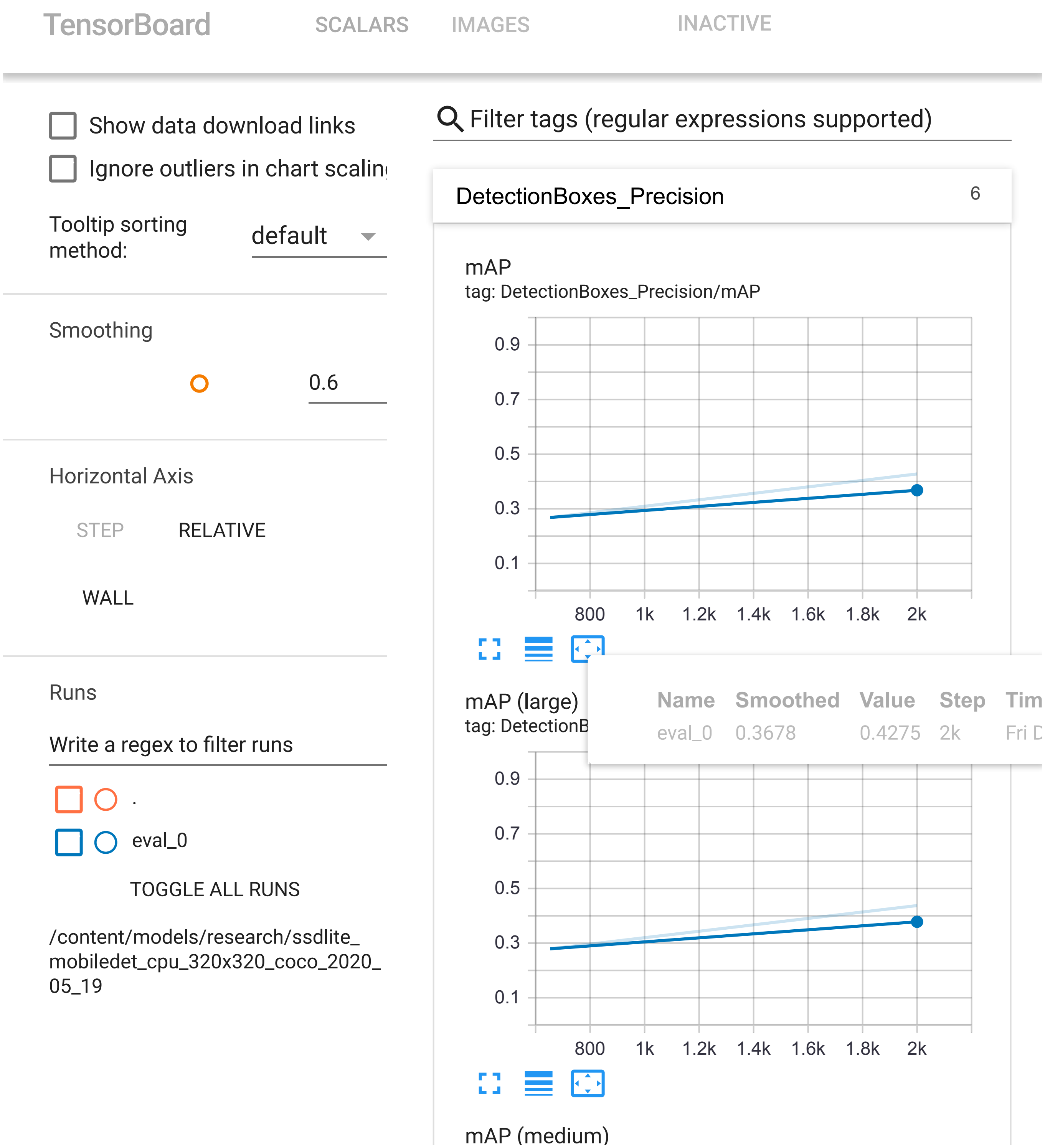
Now that we have the graph wcan convert it to TensorFlow Lite. Let's shift the runtime to TF 2. To do so, simply restart the Colab runtime.

# Optionally see the model losses in TensorBoard (within Colab Notebook)

Note If you trained for 2000 steps only you are likely to see poor numbers in TensorBoard. But as I had mentioned training a SoTA model is not the purpose of this notebook.

```
%tensorflow_version 2.x
%load_ext tensorboard
%tensorboard --logdir /content/models/research/ssdlite_mobiledet_cpu_320x320_coco_202
```

```
TensorFlow is already loaded. Please restart the runtime to change versions.
```




## Export to TFLite

```
#Imports
import tensorflow as tf
print(tf.__version__)

import os
```

```
1.15.2
```

## Quantize and serialize

For the purpose of this notebook, we will only be quantizing using the dynamic-range quantization. But you can follow this notebook if you are interested to try out the other ones like integer quantization and `float16` quantization.

As the .pb file we generated in the earlier step is a frozen graph, we need to use `tf.compat.v1.lite.TFLiteConverter.from_frozen_graph` to convert it to TFLite.

The MobileDet checkpoints we used accept 320x320 images, hence the `input_shapes` argument is specified that way. I specified the other arguments following instructions from here. **bold text**

```
model_to_be_quantized = "/content/models/research/ssdlite_mobiledet_cpu_320x320_coco_
converter = tf.compat.v1.lite.TFLiteConverter.from_frozen_graph(
    graph_def_file=model_to_be_quantized,
    input_arrays=['normalized_input_image_tensor'],
    output_arrays=['TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','T
    input_shapes={'normalized_input_image_tensor': [1, 320, 320, 3]}
)
converter.allow_custom_ops = True
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

tflite_filename = "fruits_detector" + "_dr" + ".tflite"
open(tflite_filename, 'wb').write(tflite_model)
print(f"TFLite model generated: {tflite_filename}")
!ls -lh $tflite_filename
```

```
TFLite model generated: fruits_detector_dr.tflite
-rw-r--r-- 1 root root 3.6M Dec 25 20:26 fruits_detector_dr.tflite
```

# Run inference

```
#Imports
import matplotlib
import matplotlib.pyplot as plt

import cv2
import re
import time
import numpy as np

from PIL import Image
```

## TFLite Interpreter and detection utils

Sourced from here.

```
def set_input_tensor(interpreter, image):
  """Sets the input tensor."""
  tensor_index = interpreter.get_input_details()[0]['index']
  input_tensor = interpreter.tensor(tensor_index)()[0]
  input_tensor[:, :] = image


def get_output_tensor(interpreter, index):
  """Returns the output tensor at the given index."""
  output_details = interpreter.get_output_details()[index]
  tensor = np.squeeze(interpreter.get_tensor(output_details['index']))
  return tensor


def detect_objects(interpreter, image, threshold):
  """Returns a list of detection results, each a dictionary of object info."""
  set_input_tensor(interpreter, image)
  interpreter.invoke()

  # Get all output details
  boxes = get_output_tensor(interpreter, 0)
  classes = get_output_tensor(interpreter, 1)
  scores = get_output_tensor(interpreter, 2)
  count = int(get_output_tensor(interpreter, 3))

  results = []
  for i in range(count):
    if scores[i] >= threshold:
      result = {
          'bounding_box': boxes[i],
          'class_id': classes[i],
          'score': scores[i]
      }
      results.append(result)
  return results


# Supply a path to download a relevant image
IMAGE_PATH = "https://i.ibb.co/2tsXmCV/image.png"

!wget -q -O image.png $IMAGE_PATH
Image.open('image.png')
```

```
# Load the TFLite model
interpreter = tf.lite.Interpreter(model_path="/content/fruits_detector_dr.tflite")
interpreter.allocate_tensors()
_, HEIGHT, WIDTH, _ = interpreter.get_input_details()[0]['shape']
print(f"Height and width accepted by the model: {HEIGHT, WIDTH}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
# Image preprocessing utils
def preprocess_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    original_image = img
    resized_img = tf.image.resize(img, (HEIGHT, WIDTH))
    resized_img = resized_img[tf.newaxis, :]
    return resized_img, original_image
```

```
# Define the label dictionary and color map
LABEL_DICT = {
    "orange": 1,
    "banana": 2,
    "apple": 3
}

COLORS = np.random.randint(0, 255, size=(len(LABEL_DICT), 3),
                            dtype="uint8")


# Inference utils
def display_results(image_path, threshold=0.3):
    # Load the input image and preprocess it
    preprocessed_image, original_image = preprocess_image(image_path)

    # =============Perform inference=====================
    start_time = time.monotonic()
    results = detect_objects(interpreter, preprocessed_image, threshold=threshold)
    print(f"Elapsed time: {(time.monotonic() - start_time)*1000} miliseconds")

    # =============Display the results====================
    original_numpy = original_image.numpy()
    for obj in results:
        # Convert the bounding box figures from relative coordinates
        # to absolute coordinates based on the original resolution
        ymin, xmin, ymax, xmax = obj['bounding_box']
        xmin = int(xmin * original_numpy.shape[1])
        xmax = int(xmax * original_numpy.shape[1])
        ymin = int(ymin * original_numpy.shape[0])
        ymax = int(ymax * original_numpy.shape[0])

        # Grab the class index for the current iteration
        idx = int(obj['class_id'])
        # Skip the background
        if idx >= len(LABEL_DICT):
            continue
```

```
        # Draw the bounding box and label on the image
        color = [int(c) for c in COLORS[idx]]
        cv2.rectangle(original_numpy, (xmin, ymin), (xmax, ymax),
                      color, 2)
        y = ymin - 15 if ymin - 15 > 15 else ymin + 15
        label = "{}: {:.2f}%".format(LABEL_DICT[idx],
            obj['score'] * 100)
        cv2.putText(original_numpy, label, (xmin, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # return the final imaage
    original_int = (original_numpy * 255).astype(np.uint8)
    return original_int


# Run inference and measure the inference time
resultant_image = display_results("/content/image.png", threshold=0.3)
Image.fromarray(resultant_image)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-73-a1d5f786e226> in <module>()
      1 # Run inference and measure the inference time
----> 2 resultant_image = display_results("/content/image.png", threshold=0.3)
      3 Image.fromarray(resultant_image)

  1 frames
<ipython-input-70-e89900eb19d2> in preprocess_image(image_path)
      5     img = tf.image.convert_image_dtype(img, tf.float32)
      6     original_image = img
----> 7     resized_img = tf.image.resize(img, (HEIGHT, WIDTH))
      8     resized_img = resized_img[tf.newaxis, :]
      9     return resized_img, original_image

NameError: name 'HEIGHT' is not defined
```

SEARCH STACK OVERFLOW

```
**Note** that you might see some unexpected results because the annotations in the tr


# Define the label dictionary and color map
LABEL_DICT = {
    "orange": 1,
    "banana": 2,
    "apple": 3
}

COLORS = np.random.randint(0, 255, size=(len(LABEL_DICT), 3),
                            dtype="uint8")


# Inference utils
```

```
def display_results(image_path, threshold=0.3):
    # Load the input image and preprocess it
    preprocessed_image, original_image = preprocess_image(image_path)

    # =============Perform inference=====================
    start_time = time.monotonic()
    results = detect_objects(interpreter, preprocessed_image, threshold=threshold)
    print(f"Elapsed time: {(time.monotonic() - start_time)*1000} miliseconds")

    # =============Display the results====================
    original_numpy = original_image.numpy()
    for obj in results:
        # Convert the bounding box figures from relative coordinates
        # to absolute coordinates based on the original resolution
        ymin, xmin, ymax, xmax = obj['bounding_box']
        xmin = int(xmin * original_numpy.shape[1])
        xmax = int(xmax * original_numpy.shape[1])
        ymin = int(ymin * original_numpy.shape[0])
        ymax = int(ymax * original_numpy.shape[0])

        # Grab the class index for the current iteration
        idx = int(obj['class_id'])
        # Skip the background
        if idx >= len(LABEL_DICT):
            continue

        # Draw the bounding box and label on the image
        color = [int(c) for c in COLORS[idx]]
        cv2.rectangle(original_numpy, (xmin, ymin), (xmax, ymax),
                    color, 2)
        y = ymin - 15 if ymin - 15 > 15 else ymin + 15
        label = "{}: {:.2f}%".format(LABEL_DICT[idx],
            obj['score'] * 100)
        cv2.putText(original_numpy, label, (xmin, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # return the final imaage
    original_int = (original_numpy * 255).astype(np.uint8)
    return original_int


# Run inference and measure the inference time
resultant_image = display_results("/content/image.png", threshold=0.3)
Image.fromarray(resultant_image)
```

**Note** that you might see some unexpected results because the annotations in the training dataset are faulty at places. Due to this the model training can suffer a lot.